UTTARANCHAL UNIVERSITY DEHRADUN NAAC GRADE A+

SPRING

UNIT-5

ADVANCE JAVA

TOPICS COVERED

- Basics of Spring: Spring Modules
- Spring Applications
- Spring jdbc; Jdbc template
- Example Prepared Statement
- Result Set Extractor
- Row Mapper
- Named Parameter
- Simple jdbc Template
- Mvc in Spring

SPRING - INTRODUCTION

- Spring framework is an open source Java platform that provides comprehensive infrastructure support for developing robust Java applications very easily and rapidly.
- It was initially written by **Rod Johnson** and was first released under the Apache 2.0 license in June 2003.
- Spring is lightweight when it comes to size and transparency. The basic version of Spring framework is around 2MB.
- Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications
- Spring handles the infrastructure so the programmer focus on the application.

SPRING - INTRODUCTION

- Spring enables you to build applications from “plain old Java objects” (POJOs) and to apply enterprise services non-invasively to POJOs.
- This capability applies to the Java SE programming model and to full and partial Java EE.
- It can be thought of as a ***framework of frameworks*** because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF, etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.
- The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc.

BENEFITS OF SPRING

- Spring allows developers to create high-quality applications using POJOs. Using POJOs exclusively has the advantage of not requiring an EJB container like an app server, allowing for the use of a resilient servlet container like Tomcat or a different commercial option.
- Spring is structured in a modular way. Despite the large quantity of packages and classes, you only need to focus on the ones you require and disregard the others.
- Spring leverages existing technologies such as ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, and other view technologies rather than creating new ones.
- It is easy to test an application developed in Spring since any code specific to the environment is placed within the framework. Additionally, employing JavaBeanstyle POJOs makes it simpler to utilize dependency injection for inserting test data.

BENEFITS OF SPRING

- Spring's web framework is a well-structured web MVC framework that offers a strong alternative to web frameworks like Struts or other complex or less popular options.
- Spring offers a user-friendly interface for converting technology-specific exceptions (such as those raised by JDBC, Hibernate, or JDO) into uniform, unchecked exceptions.
- Small IoC containers are typically lighter than EJB containers, especially in comparison. This is advantageous for developing and deploying applications on computers with restricted memory and CPU resources.
- Spring offers a transaction management interface that is versatile enough to handle both small-scale transactions with a single database and large-scale transactions with JTA.

DEPENDENCY INJECTION (DI)

- The technology that Spring is most identified with is the **Dependency Injection (DI)** flavour of Inversion of Control.
- The **Inversion of Control (IoC)** is a general concept, and it can be expressed in many different ways.
- Dependency Injection is merely one concrete example of Inversion of Control.
- When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while unit testing.
- Dependency Injection helps in gluing these classes together and at the same time keeping them independent.

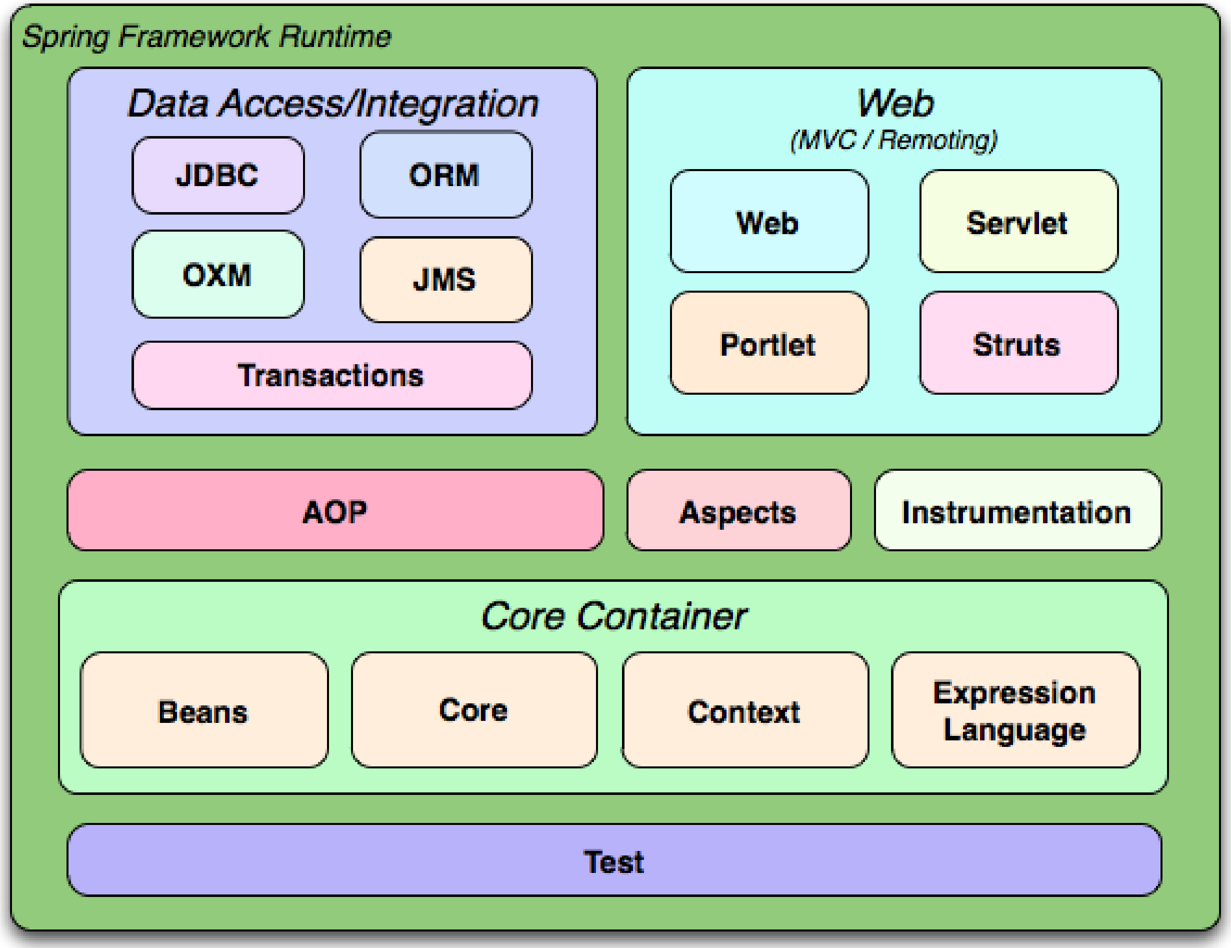
DEPENDENCY INJECTION (DI)

- The dependency part translates into an association between two classes.
- For example, class A is dependent of class B. Now, let's look at the second part, injection. All this means is, class B will get injected into class A by the IoC.
- Dependency injection can happen in the way of passing parameters to the constructor or by post-construction using setter methods.

ASPECT ORIENTED PROGRAMMING (AOP)

- An essential element of Spring is the Aspect Oriented Programming (AOP) framework.
- Cross-cutting concerns, which encompass various points within an application, are distinct from the business logic of the application.
- Logging, declarative transactions, security, caching, etc. are all examples of common good aspects.
- In object-oriented programming, classes are the main unit of modularity, while in aspect-oriented programming, aspects serve as the main unit of modularity.
- DI assists in separating application objects from one another, whereas AOP aids in separating cross-cutting concerns from the affected objects.
- The AOP module in the Spring Framework enables aspect-oriented programming by defining method-interceptors and pointcuts to separate code implementing functionality.

SPRING MODULES



SPRING MODULES

- Spring could potentially be a one-stop shop for all enterprise applications.
- Spring is modular, allowing you to pick and choose which modules are applicable to you, without having to bring in the rest.
- The Spring Framework provides about 20 modules which can be used based on an application requirement.
- These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, and Test

UTTARANCHAL UNIVERSITY DEHRADUN NAAC GRADE A+

MODULE: CORE CONTAINER

- The Core Container consists of the Core, Beans, Context, and Expression Language modules:
- The **Core** module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
- The **Bean** module provides BeanFactory, which is a sophisticated implementation of the factory pattern.
- The **Context** module expands on the strong foundation of the Core and Beans modules, serving as a gateway to interact with any defined and configured objects. The ApplicationContext interface is the central focus of the Context module.
- The **SpEL** module offers a robust expression language for dynamic querying and manipulation of an object graph.

MODULE: DATA ACCESS/INTEGRATION

- The Data Access/Integration layer is comprised of the JDBC, ORM, OXM, JMS, and Transaction modules, with the following explanation for each one.
- The **JDBC** module offers a JDBC-abstraction layer that eliminates the necessity for laborious JDBC coding.
- The **ORM** module offers connectivity layers for commonly used object-relational mapping APIs such as JPA, JDO, Hibernate, and iBatis.
- The **OXM** module offers a layer of abstraction that backs Object/XML mapping implementations such as JAXB, Castor, XMLBeans, JiBX, and XStream.
- The **JMS** module in Java Messaging Service includes functionalities for creating and receiving messages.
- The **Transaction** module offers both programmatic and declarative transaction management for classes with special interfaces and for all POJOs.

MODULE: WEB

- The Web layer consists of the Web, Web-MVC, Web-Socket, and Web-Portlet modules which are as follows −
- The **Web** module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
- The **Web-MVC** module contains Spring's Model-View-Controller (MVC) implementation for web applications.
- The **Web-Socket** module provides support for WebSocket-based, two-way communication between the client and the server in web applications.
- The **Web-Portlet** module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

MODULE: MISCELLANEOUS

- There are few other important modules like AOP, Aspects, Instrumentation, Web and Test modules which are as follows −
- The **AOP** module provides an aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.
- The **Aspects** module provides integration with AspectJ, which is again a powerful and mature AOP framework.
- The **Instrumentation** module provides class instrumentation support and class loader implementations to be used in certain application servers.
- The **Messaging** module provides support for STOMP as the WebSocket sub-protocol to use in applications. It also supports an annotation programming model for routing and processing STOMP messages from WebSocket clients.
- The **Test** module supports the testing of Spring components with JUnit or TestNG frameworks.

SPRING JDBC FRAMEWORK

- Spring JDBC Framework manages all the low-level aspects, including opening the connection, preparing and executing SQL statements, handling exceptions, managing transactions, and ultimately closing the connection.
- Spring JDBC offers multiple methods and various classes to connect with the database.
- This is the primary framework class that oversees all database interactions and exception management.

JDBCTEMPLATE CLASS

- The JDBC Template class executes SQL queries, updates statements, stores procedure calls, performs iteration over ResultSets, and extracts returned parameter values.
- It also catches JDBC exceptions and translates them to the generic, more informative, exception hierarchy defined in the org.springframework.dao package.
- Instances of the JdbcTemplate class are threadsafe once configured. So you can configure a single instance of a JdbcTemplate and then safely inject this shared reference into multiple DAOs.
- A common practice when using the JDBC Template class is to configure a DataSource in your Spring configuration file, and then dependency-inject that shared DataSource bean into your DAO classes, and the JdbcTemplate is created in the setter for the DataSource.

CONFIGURING DATA SOURCE

- Let us create a database table Student in our database TEST.
- We assume you are working with MySQL database, if you work with any other database then you can change your DDL and SQL queries accordingly.

```
CREATE TABLE Student(
   ID   INT NOT NULL AUTO_INCREMENT,
   NAME VARCHAR(20) NOT NULL,
   AGE  INT NOT NULL,
   PRIMARY KEY (ID)
);
```

UTTARANCHAL UNIVERSITY DEHRADUN NAAC GRADE A+

CONFIGURING DATA SOURCE

- Now we need to supply a DataSource to the JDBC Template so it can configure itself to get database access.
- You can configure the DataSource in the XML file with a piece of code as shown in the following code snippet

```
<bean id = "dataSource"
   class = "org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name = "driverClassName" value = "com.mysql.jdbc.Driver"/>
   <property name = "url" value = "jdbc:mysql://localhost:3306/TEST"/>
   <property name = "username" value = "root"/>
   <property name = "password" value = "password"/>
</bean>
```

DATA ACCESS OBJECT (DAO)

- DAO stands for Data Access Object, which is commonly used for database interaction.
- DAOs exist to provide a means to read and write data to the database and they should expose this functionality through an interface by which the rest of the application will access them.
- The DAO support in Spring makes it easy to work with data access technologies like JDBC, Hibernate, JPA, or JDO in a consistent way.

EXECUTING SQL STATEMENTS

- **Querying for an integer**

```
String SQL = "select count(*) from Student";
int rowCount = jdbcTemplateObject.queryForInt( SQL );
```

- **Querying for a long**

```
String SQL = "select count(*) from Student";
long rowCount = jdbcTemplateObject.queryForLong( SQL );
```

EXECUTING SQL STATEMENTS

- **A simple query using a bind variable**

```
String SQL = "select age from Student where id = ?";
int age = jdbcTemplateObject.queryForInt(SQL, new Object[]{10});
```

- **Querying for a String**

```
String SQL = "select name from Student where id = ?";
String name = jdbcTemplateObject.queryForObject(SQL, new Object[]{10}, String.class);
```

- **Querying and returning an object**

```
String SQL = "select * from Student where id = ?";
Student student = jdbcTemplateObject.queryForObject(   SQL, new Object[]{10}, new StudentMapper());
public class StudentMapper implements RowMapper<Student> {
  public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
    Student student = new Student();
    student.setID(rs.getInt("id"));
    student.setName(rs.getString("name"));
    student.setAge(rs.getInt("age"));
    return student;
  }
}
```

QUERYING AND RETURNING MULTIPLE OBJECTS

```
String SQL = "select * from Student";
List<Student> students = jdbcTemplateObject.query(  SQL, new StudentMapper());
public class StudentMapper implements RowMapper<Student> {
  public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
    Student student = new Student();
    student.setID(rs.getInt("id"));
    student.setName(rs.getString("name"));
    student.setAge(rs.getInt("age"));

    return student;
  }  }
```

QUERIES WITH OBJECTS

- **Inserting a row into the table**

```
String SQL = "insert into Student (name, age) values (?, ?)";
jdbcTemplateObject.update( SQL, new Object[]{"Zara", 11} );
```

- **Updating a row into the table**

```
String SQL = "update Student set name = ? where id = ?";
jdbcTemplateObject.update( SQL, new Object[]{"Zara", 10} );
```

- **Deleting a row from the table**

```
String SQL = "delete Student where id = ?";
jdbcTemplateObject.update( SQL, new Object[]{20} );
```

PREPARED STATEMENT

- The **org.springframework.jdbc.core.PreparedStatementSetter** interface acts as a general callback interface used by the JdbcTemplate class.
- This interface sets values on a PreparedStatement provided by the JdbcTemplate class, for each of a number of updates in a batch using the same SQL.
- Implementations are responsible for setting any necessary parameters.
- It's easier to use this interface than PreparedStatementCreator.
- The JdbcTemplate will create the PreparedStatement, with the callback only being responsible for setting parameter values.

PREPARED STATEMENT

- **Interface Declaration**
- Following is the declaration for org.springframework.jdbc.core.PreparedStatementSetter interface –
- **public interface PreparedStatementSetter**
- **Usage**
- Step 1 – Create a JdbcTemplate object using a configured datasource.
- Step 2 – Use JdbcTemplate object methods to make database operations while passing PreparedStatementSetter object to replace place holders in query.

PREPARED STATEMENT--SYNTAX

```
final String SQL = "select * from Student where id = ? ";
List <Student> students = jdbcTemplateObject.query(
  SQL, new PreparedStatementSetter() {

  public void setValues(PreparedStatement preparedStatement) throws
SQLException {
     preparedStatement.setInt(1, id);
  }
},
new StudentMapper());
```

SPRING JDBC - RESULTSETEXTRACTOR INTERFACE

- The **org.springframework.jdbc.core.ResultSetExtractor** interface is a callback interface used by JdbcTemplate's query methods.
- Implementations of this interface perform the actual work of extracting results from a ResultSet, but don't need to worry about exception handling.
- SQLExceptions will be caught and handled by the calling JdbcTemplate.
- This interface is mainly used within the JDBC framework itself.
- **A RowMapper is usually a simpler choice for ResultSet processing, mapping one result object per row instead of one result object for the entire ResultSet.**

SPRING JDBC - RESULTSETEXTRACTOR INTERFACE

- **Interface Declaration**
- Following is the declaration for **org.springframework.jdbc.core.ResultSetExtractor** interface –
- public interface ResultSetExtractor

- **Usage**
- Step 1 – Create a JdbcTemplate object using a configured datasource.
- Step 2 – Use JdbcTemplate object methods to make database operations while parsing the resultset using ResultSetExtractor.

SPRING JDBC - RESULTSETEXTRACTOR INTERFACE

```
public List<Student> listStudents() {
  String SQL = "select * from Student";
  List <Student> students =
jdbcTemplateObject.query(
    SQL, new
ResultSetExtractor<List<Student>>(){
      public List<Student> extractData(ResultSet
rs) throws SQLException, DataAccessException {
         List<Student> list = new ArrayList<Student>();
         while(rs.next()){
           Student student = new Student();
 student.setId(rs.getInt("id"));
student.setName(rs.getString("name"));
student.setAge(rs.getInt("age"));

student.setDescription(rs.getString("description"));
              student.setImage(rs.getBytes("image"));
              list.add(student);
             }
             return list;
          }        }     );
     return students;
}
```

SPRING JDBC - ROWMAPPER INTERFACE

- The **org.springframework.jdbc.core.RowMapper<T>** interface is used by JdbcTemplate for mapping rows of a ResultSet on a per-row basis.
- Implementations of this interface perform the actual work of mapping each row to a result object.
- SQLExceptions if any thrown will be caught and handled by the calling JdbcTemplate.
- Following is the declaration for **org.springframework.jdbc.core.RowMapper<T>** interface

```
public interface RowMapper<T>
```

SPRING JDBC - ROWMAPPER INTERFACE

- **Syntax**

```
String SQL = "select * from Student";
List <Student> students = jdbcTemplateObject.query(SQL, new
StudentMapper());
```

- Where
- SQL – Read query to read all student records.
- jdbcTemplateObject – StudentJDBCTemplate object to read student records from database.
- StudentMapper – StudentMapper object to map student records to student objects.

SPRING JDBC - NAMEDPARAMETERJDBCTEMPLATE CLASS

- The **org.springframework.jdbc.core.NamedParameterJdbcTemplate** class is a template class with a basic set of JDBC operations, allowing the use of named parameters rather than traditional '?' placeholders.
- This class delegates to a wrapped JdbcTemplate once the substitution from named parameters to JDBC style '?' placeholders is done at execution time.
- It also allows to expand a list of values to the appropriate number of placeholders.
-

Following is the declaration for **org.springframework.jdbc.core.NamedParameterJdbcTemplate class**

```
public class NamedParameterJdbcTemplate  extends Object
   implements NamedParameterJdbcOperations
```

SPRING JDBC - NAMEDPARAMETERJDBCTEMPLATE CLASS

```
MapSqlParameterSource in = new MapSqlParameterSource();

in.addValue("id", id);

in.addValue("description",  new SqlLobValue(description, new DefaultLobHandler()),
Types.CLOB);

String SQL = "update Student set description = :description where id = :id";

NamedParameterJdbcTemplate jdbcTemplateObject = new
NamedParameterJdbcTemplate(dataSource);

jdbcTemplateObject.update(SQL, in);
```

- Where,

in – SqlParameterSource object to pass a parameter to update a query.
SqlLobValue – Object to represent an SQL BLOB/CLOB value parameter.
jdbcTemplateObject – NamedParameterJdbcTemplate object to update student object in the database.

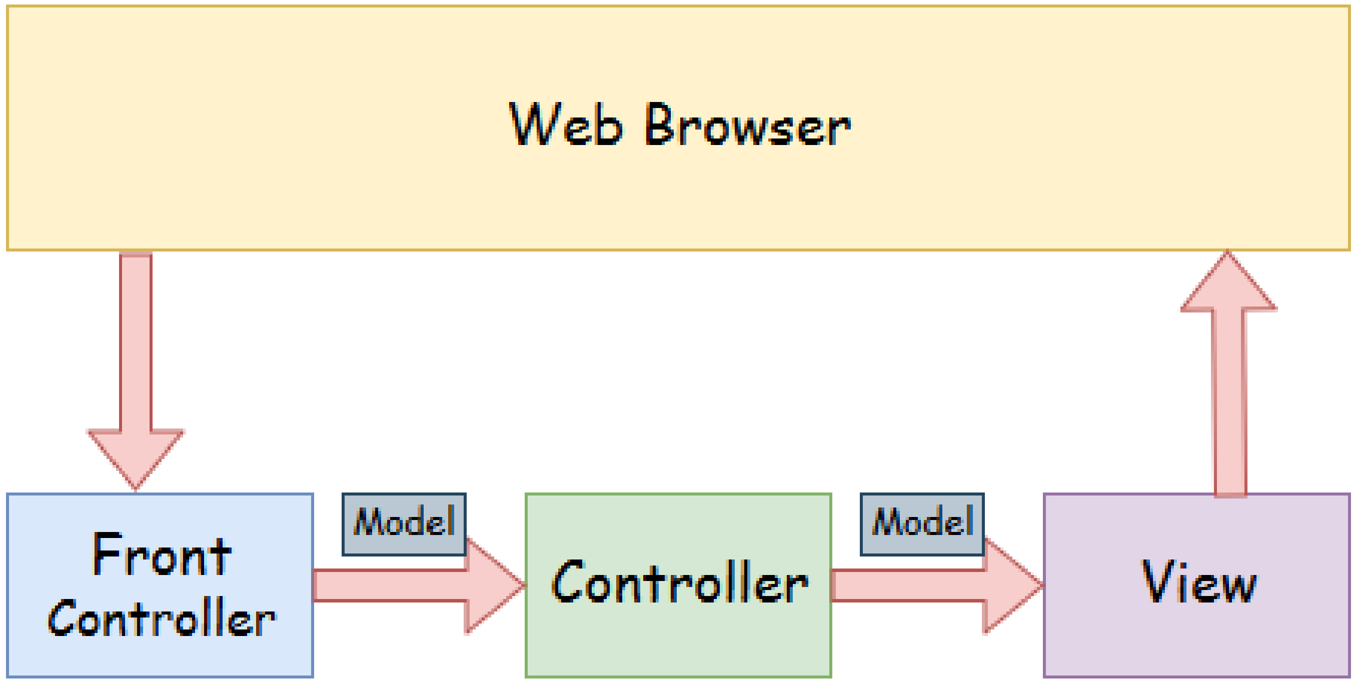
MVC IN SPRING

SPRING MVC

- A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern.
- It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.
- A Spring MVC provides an elegant solution to use MVC in spring framework by the help of **DispatcherServlet**.
- Here, **DispatcherServlet** is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.

UTTARANCHAL UNIVERSITY DEHRADUN NAAC GRADE A+

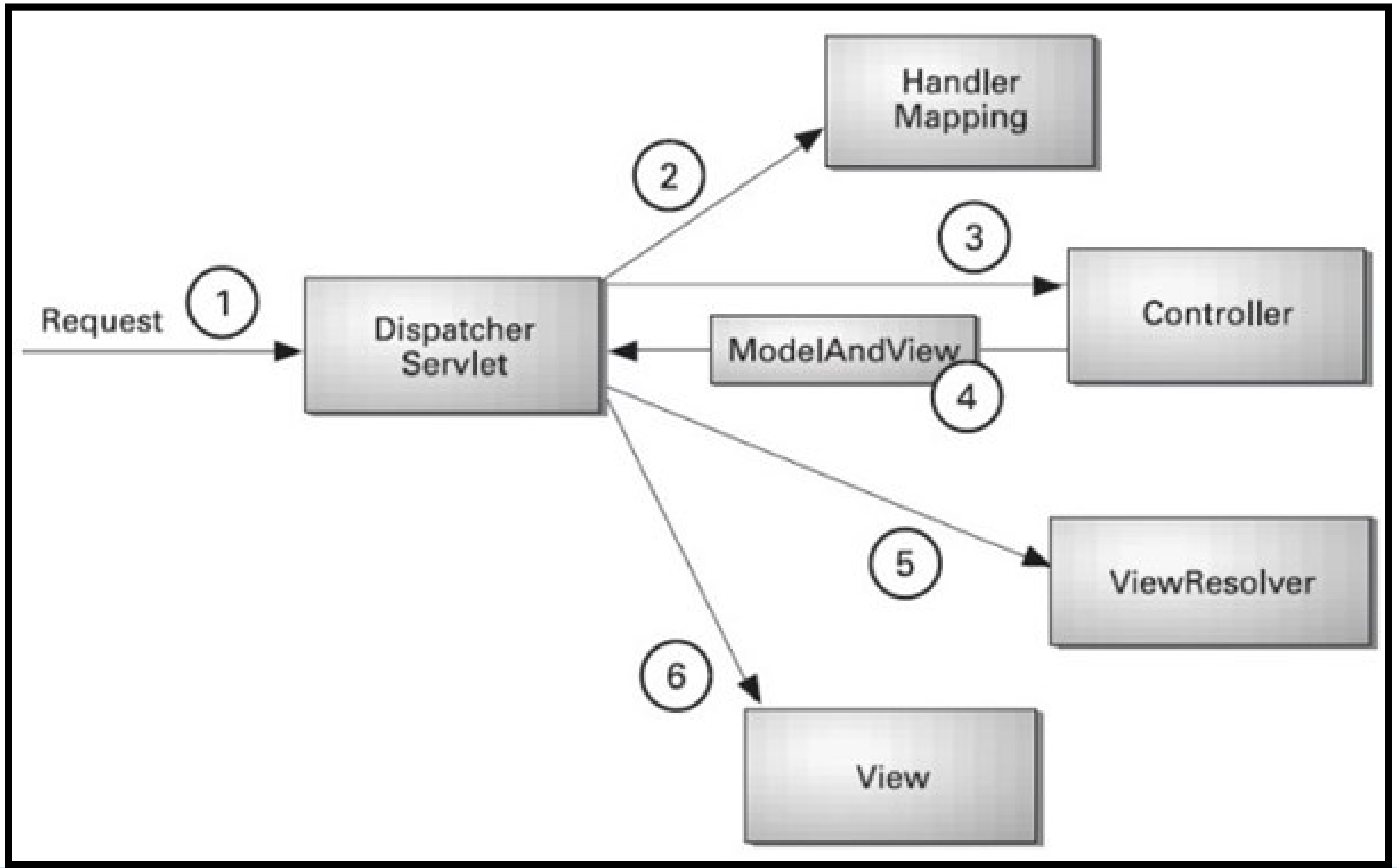
SPRING MVC



SPRING MVC

- **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
- **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
- **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
- **Front Controller** - In Spring Web MVC, the Dispatcher Servlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.

UNDERSTANDING THE FLOW OF SPRING WEB MVC



UNDERSTANDING THE FLOW OF SPRING WEB MVC

- All incoming requests are captured by the DispatcherServlet, which serves as the main controller, as shown in the diagram.
- The DispatcherServlet retrieves a handler mapping entry from the XML file and then directs the request to the controller.
- The Model And View object is returned by the controller.
- The DispatcherServlet verifies the presence of a view resolver entry in the XML file and calls the designated view component.

ADVANTAGES OF SPRING MVC FRAMEWORK

- **Separate roles** -The Spring MVC divides each function, with specialized objects like model object, controller, command object, view resolver, DispatcherServlet, validator, etc. fulfilling each role.
- **Lightweight** - It utilizes a light-weight servlet container for building and deploying your application.
- **Powerful configuration**-It offers a solid configuration for framework and application classes with seamless referencing between contexts, like from web controllers to business objects and validators.
- **Rapid development**-Spring MVC allows for rapid development in a parallel manner.

ADVANTAGES OF SPRING MVC FRAMEWORK

- **Reusable business code-** refers to the practice of utilizing the already existing business objects rather than creating new ones.
- **Easy to test -** Typically in Spring, we generate JavaBeans classes that allow you to provide test data through the setter methods.
- **Flexible Mapping-**It offers precise annotations for seamlessly redirecting the page.

SPRING WEB MVC FRAMEWORK EXAMPLE

- Load the spring jar files or add dependencies in the case of Maven
- Create the controller class
- Provide the entry of controller in the web.xml file
- Define the bean in the separate XML file
- Display the message in the JSP page
- Start the server and deploy the project

REQUIRED JAR FILES OR MAVEN DEPENDENCY

- To run this example, you need to load:
- Spring Core jar files
- Spring Web jar files
- JSP + JSTL jar files (If you are using any another view technology then load the corresponding jar files).

1. Provide project information and configuration in the pom.xml file.
2. Create the controller class

```
package com.welcome;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class HelloController {
@RequestMapping("/")
   public String display()
   {
      return "index";
   }   }
```

3. Provide the entry of controller in the web.xml file

- In this xml file, we are specifying the servlet class DispatcherServlet that acts as the front controller in Spring Web MVC. All the incoming request for the html file will be forwarded to the DispatcherServlet.

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.c
om/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_3_0.xsd" id="WebApp_ID" version="3.0">

  <display-name>SpringMVC</display-name>

   <servlet>

    <servlet-name>spring</servlet-name>

    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
</web-app>
```

4. DEFINE THE BEAN IN THE XML FILE SPRING-SERVLET.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
```

4. DEFINE THE BEAN IN THE XML FILE

SPRING-SERVLET.XML

```xml
http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd">
<!-- Provide support for component scanning -->
  <context:component-scan base-package="com.welcome" />
   <!--Provide support for conversion, formatting and validation -->
  <mvc:annotation-driven/>
 </beans>
```

UTTARANCHAL UNIVERSITY DEHRADUN NAAC GRADE A+

5. DISPLAY THE MESSAGE IN THE JSP PAGE

- **Index.jsp**

```
<html>

<body>

<p>Welcome to Spring MVC </p>

</body>

</html>
```